New in CoCoA-5.2.2 and CoCoALib-0.99560 for SC-Square

John Abbott *[1] and Anna M. Bigatti[2]

[1] Institut für Mathematik, Universität Kassel, Germany
[2] Dipartimento di Matematica, Università degli Studi di Genova, Italy

Abstract. CoCoA-5 is an interactive Computer Algebra System for Computations in Commutative Algebra, particularly Gröbner bases. It offers a dedicated, mathematician-friendly programming language, with many built-in functions. Its mathematical core is CoCoALib, an open-source C++ library, designed to facilitate integration with other software.
We give an overview of the latest developments of the library and of the system, in particular relating to the project SC-Square.

1 Introduction

We briefly recall what is described in [2]. The CoCoA project dates back to 1987, with the first public release of its interactive system in 1989. The aim has always been to offer a convenient software laboratory for studying Computational Commutative Algebra, specifically ideals of multivariate polynomials (*e.g.* Gröbner bases).

In the last few years the CoCoA software has undergone a profound change in its internal design: its "mathematical expertise" resides in the C++ software library [3]; the interactive system [7] uses an interpreter which grants easy access to CoCoALib's capabilities. All code is free and open source (licence GPL-3).

We give an overview of the latest developments of the library and of the system (release May 2017). There are some aspects of particular interest to the project SC^2. One is the efficient computation of the *minimal polynomial* (Section 3.2) which is the main tool for many operations on 0-dimensional ideals, and also for studying polynomial systems. Another aspect is first prototypes in communication between CoCoA and MathSAT (Section 3.3).

2 News about the overall design

The design of CoCoALib and CoCoA-5 is obviously quite stable (because of their size and age), so naturally the main changes are the addition of numerous new functions. There are, nevertheless, a few handy changes in the *user experience*:

- **verbose mode** for printing "internal progress information";
- clean **interrupt handling** (including a "timeout" capability).

* J. Abbott is an INdAM-COFUND Marie Curie Fellow

2.1 Verbose mode

The new function `SetVerbosityLevel(N)` sets the global verbosity level to `N`. Various functions defined in CoCoALib and in the CoCoA packages print out some internal progress messages when the global verbosity level is higher than a certain threshold value. Higher levels trigger greater verbosity, and the manual entries for each function list the relevant thresholds. This is useful to developers who want to understand what is happening inside important functions (and also to the curious).

For instance, the lowest level giving information on the progress of Gröbner bases is 100: everytime a new polynomial is found, a line like this is printed:

```
myDoGBasis[1]: --New poly in GB: len(GB) = 5 len(pairs) = 2
```

Besides `GBasis`, verbose information can be produced by numerous other functions: for instance, those described in Section 3.2. Indeed, the number of functions which respond to the verbosity setting is steadily increasing — details are in the documentation (*e.g.* in CoCoA-5 type "`?verbose`").

Since Gröbner Basis computation is quite pervasive in CoCoA its verbosity threshold level is quite high, whereas in general other functions have levels in the range 10–99. The values 1-9 are effectively reserved for user defined functions, so that they may be used without triggering any verbosity from CoCoA.

2.2 Interrupt handling

In some hard Gröbner basis computations, by setting the verbosity level, one may see that the number of pairs yet to be processed is unfeasibly high. The user may then choose to interrupt the computation by typing `Ctrl-C`: the computation will be interrupted as soon as the reduction of the current S-polynomial terminates. The user may instead set a *timeout*: the calculation is automatically interrupted after the specified computation time.

This interruption cancels the incomplete Gröbner basis computation, and returns the computer to the state it was in just before the Gröbner basis computation was begun. Doing this correctly is not entirely trivial: thanks to its clean, exception-safe design, CoCoALib guarantees that no memory has been lost, and no data has been modified.

The design for doing it is quite simple: it is enough to add a call to the CoCoALib function `CheckForInterrupt` in potentially critical loops: this function checks if the "interrupt flag" has been set, and if so, throws an exception. Thus it may be used in any function in CoCoALib (even those defined by users of CoCoALib, if they have been written in an exception-safe way!), and the number of interruptible CoCoALib functions is gradually increasing.

On the other side, all code written in the interpreted CoCoA language is intrinsically interruptible, with no addition to the code, thanks to the design of the interpreter of CoCoA-5.

A small aside: one might like to interrupt a computation, perform some checks and then decide whether to continue the computation from that stage. This option is not (yet) implemented.

2.3 Removing old features

As software develops, sometimes a few old functions become obsolete because a better design had evolved. For some users this fact is a deterrent to updating to the latest software release, for fear that their programs would stop working.

In CoCoA-5 and CoCoAlib we have opted to first declare such functions *obsolescent*, *i.e.* they will become *obsolete* in a few years time. We put them into the special files, `obsolescent.cpkg5` for CoCoA-5, and `obsolescent.[HC]` for CoCoALib. They still work, but print out a message warning about obsolescence but also giving useful guidance on the new function to call instead. We use this approach to allow users time, and provide help, to update their programs.

3 News about CoCoA-5 and CoCoALib

A list of all new functions and features in CoCoA-5 can always be found at the CoCoA-5 download page, or obtained by running the function `RelNotes()`. In addition, a fully detailed description of the work and decisions made for each release of CoCoA-5 and CoCoALib may be found at the CoCoA *Redmine* website `https://cocoa.dima.unige.it/redmine`.

Most of the new functions added to CoCoA-5 have actually been added to CoCoALib, and then made available to CoCoA-5; the remaining few will be ported to CoCoALib shortly.

Here we mention some of the more significant additions.

- several efficient operations specifically for 0-dimensional ideals: (Section 3.2)
  - `MinPolyQuot`
  - `PrimaryDecomposition`
  - `IsMaximal`, `IsPrimary`, `IsRadical`, `radical`
  - `FrobeniusMat`, `FrobeniusBasis`
  - `QuotientBasisSorted`
- Algebraic extensions and `FactorAlgExt`
- `rgin` — generic initial ideal wrt. `DegRevLex` ordering (regardless of the ordering associated to the polynomial ring)
- `IdealOfProjectivePoints` — the optimized C code which was written for CoCoA-4 (see [4]) is now accessible from CoCoALib and CoCoA-5
- more convenient access to Gröbner bases in the Gröbner fan (*e.g.* via the functions `GroebnerFanIdeals` and `CallOnGroebnerFanIdeals`)
- access to several functions of the `Gfan` library [11]: for example `GFanContainsPositiveVector`, `GFanGeneratorsOfLinealitySpace`,...
- `BettiNumbers` — for both standard and multigraded resolutions
- `StagedTrees` — a surprising application of monomial ideals to the investigation of putative causal interpretations of datasets in statistics (see [10])
- faster conversion from a `string` to a `RingElem` — this is now the best way to read a large polynomial with very many terms

3.1 News specific for CoCoALib

Special effort has always been invested in making the CoCoALib code clean and portable; it now has a more portable configure-build-install procedure.

CoCoALib is currently written using standard C++03; this does mean that compilation with a C++11 compiler may produce some harmless warnings about "deprecated" features. A proper update to a newer C++ standard is imminent.

3.2 Minimal polynomials

It is well-known that if K is a field and R is a zero-dimensional affine K-algebra, *i.e.* a zero-dimensional algebra of type $R = K[x_1, \ldots, x_n]/I$, then R is a finite-dimensional K-vector space. Consequently, it is not surprising that minimal and characteristic polynomials from Linear Algebra can be successfully used to detect properties of R. For example, in the quotient ring $R = \mathbb{Q}[x, y]/(x^2, y^2)$ the minimal polynomial for the residue class of $x + y$ is z^3.

This point of view was taken systematically in the book [12] where the particular importance of minimal polynomials (rather greater than that of characteristic polynomials) emerged quite clearly. We studied the theory of minimal polynomials, developing efficient algorithms to compute them, and finding practical and efficient applications; here we recall the main results we described in [5] and in [13].

The first step was to implement in CoCoALib algorithms for computing the minimal polynomial of an element of R and of a K-endomorphism of R (functions `MinPolyQuotElim`, `MinPolyQuotMat`, `MinPolyQuotDef`).

For computing minimal polynomials of elements of an algebra over $\mathbb{Q}$ we use a modular approach and the rational reconstruction described in [1]. As always, various obstacles for termination and correctness had to be overcome. In particular, we deal with the notion of *reduction of an ideal modulo p*, introducing the related notions of *ugly, usable, good and bad primes*. This is further investigated in a forthcoming paper [6].

After having made the computation of minimal polynomials quite swift, we then showed how they can be successfully and efficiently used to compute several important invariants of zero-dimensional affine K-algebras. More specifically, we described algorithms which determine whether a zero-dimensional ideal is radical, maximal or primary, and another to compute the radical of a zero-dimensional ideal. (In particular, it is noteworthy that in the case of coefficient fields of small finite characteristic, Frobenius spaces play a fundamental role, as described in [12, 13]). Finally, we described how to compute the primary decomposition of a zero-dimensional affine K-algebra.

As we said, all the algorithms described have been implemented in CoCoA. Their merits are illustrated by the tables of timings in [5].

The efficient computation of minimal polynomials is also a key ingredient in solving polynomial systems, and together with the other derived algorithms we believe we can develop good tools for the SC^2community.

3.3 Communication with MathSAT

The philosophy behind CoCoALib prizes ease of use for humans, but also ease of communication with other systems, as already mentioned in [2]. There is a "symbiotic" integration with Normaliz [9], a system specialized in computations on affine monoids, vector configurations, lattice polytopes, and rational cones. CoCoALib and CoCoA-5 have functions calling *libnormaliz* (see [8]), while *Nmz-Integrate* uses CoCoALib for multivariate polynomial arithmetic.

The first few steps along these lines have recently been made between CoCoALib and MathSAT in strict collaboration with Alberto Griggio. In particular, some practical aspects were discussed, outlining these directions of development:

1. MathSAT using CoCoA's `RingTwinFloat` arithmetic (instead of arbitrary precision rationals, GMP `mpq`);
2. Making a collection of systems of polynomial equalities and inequalities arising from typical MathSAT computations, to be solved by CoCoALib and CoCoA-5;
3. enabling CoCoALib to call some of MathSAT's functions;
4. CoCoA-5 offering a handy interface for MathSAT through CoCoALib;
5. MathSAT using CoCoALib for solving systems of polynomial inequalities.

The development work since then has reached the following stage:

1. We have a working prototype using just `RingTwinFloat` which appeared to make MathSAT faster in those (rare) cases where MathSAT needs very high precision: MathSAT currently uses a "double arithmetic" adapting to the computations by using, as necessary, machine integers or GMP `mpq`; it is clearly unbeatable when using machine arithmetic, but it's worth further testing for the cases needing high precision.
2. We made a first collection of benchmark examples, and tested some of them in CoCoA-5. Inequalities are not trivially dealt with by CoCoA's algorithms, but we got very encouraging results in detecting `unsat`, due to the very special structure of the examples tested. We now need to design how to make the conversion to CoCoA-5 of these examples (more) automatic, providing a wide spectrum of cases to test.
3. The first prototype of calling MathSAT through CoCoALib is in the example `ex-MathSat1` calling `msat_solve`. This is already part of the CoCoALib distribution from version 0.99550 (released in May 2017). A more evolved interface, with a proper CoCoALib C++ wrapper class for the data type `msat_env` is part of the forthcoming CoCoALib 0.99560 distribution (September 2017). It still needs some common development with MathSAT to minimize the overhead of data conversion.
4. The first prototype of calling MathSAT functionalities via CoCoA-5 is the function `MSatLinSolve`. This is part of the CoCoA-5.2.2 distribution (based on CoCoALib 0.99560, September 2017). Type `? MathSAT` in CoCoA-5 for the latest news and examples.

5. *Future*. With the work about minimal polynomials (Section 3.2) CoCoALib has a new tool for tackling systems of polynomial equations, but inequalities are not (yet) in its nature: the examples from item 2 will inspire new investigations for this SC-Square goal.

References

1. J. Abbott, *Fault-Tolerant Modular Reconstruction of Rational Numbers*, J. Symb. Comp. 80P3, pp. 707–718, 2017.
2. J. Abbott, A.M. Bigatti, *CoCoA and CoCoALib: Fast prototyping and flexible C++ library for Computations in Commutative Algebra* Proc. 1st Workshop on Satisfiability Checking and Symbolic Computation, SC-Square 2016, CEUR Workshop Proceedings 1804, pp. 1–3, 2016.
3. J. Abbott, A.M. Bigatti, *CoCoALib: a C++ library for doing Computations in Commutative Algebra* Available at `http://cocoa.dima.unige.it/cocoalib`
4. J. Abbott, A.M. Bigatti, M. Kreuzer, L. Robbiano, *Computing Ideals of Points*, Journal of Symbolic Computation 30, 2000, pp. 341–356.
5. J. Abbott, A. Bigatti, E. Palezzato, L. Robbiano, *Computing and Using Minimal Polynomials*, `arXiv:1704.03680`, 2017.
6. J. Abbott, A. Bigatti, L. Robbiano, *Ideals modulo p*, In preparation.
7. J. Abbott, A.M. Bigatti, G. Lagorio *CoCoA-5: a system for doing Computations in Commutative Algebra* Available at `http://cocoa.dima.unige.it`
8. J. Abbott, A.M. Bigatti, C. Söger, *Integration of libnormaliz in CoCoALib and CoCoA 5* Proc. ICMS 2014, Springer LNCS 8592, pp. 647–653, 2014.
9. W. Bruns, B. Ichim, T. Römer, R. Sieg, C. Söger *Normaliz. Algorithms for rational cones and affine monoids.* Available from `http://www.normaliz.uni-osnabrueck.de`
10. C. Görgen, A.M. Bigatti, E. Riccomagno, J. Smith *Discovery of statistical equivalence classes using computer algebra* `arXiv:1705.09457`
11. A.N. Jensen, *Gfan, a software system for Gröbner fans and tropical varieties*. Available from `http://home.math.au.dk/jensen/software/gfan/gfan.html`
12. M. Kreuzer and L. Robbiano, *Computational Linear and Commutative Algebra*, Springer, Heidelberg 2016.
13. E. Palezzato, *Minimal Polynomials, Sectional Matrices, and Applications*; PhD. thesis, Università degli Studi di Genova, 2017.